Simulator Semantics for System Level Formal Verification

Toni Mancini Federico Mari Annalisa Massini Igor Melatti Enrico Tronci
Computer Science Department - Sapienza University of Rome
Via Salaria 113, I-00198 Roma, Italy
{tmancini,mari,massini,melatti,tronci}@di.uniroma1.it

Many simulation based *Bounded Model Checking* approaches to System Level Formal Verification (SLFV) have been devised. Typically such approaches exploit the capability of simulators to save computation time by saving and restoring the state of the system under simulation. However, even though such approaches aim to (bounded) formal verification, as a matter of fact, the simulator behaviour is not formally modelled and the proof of correctness of the proposed approaches basically relies on the intuitive notion of simulator behaviour. This gap makes it hard to check if the optimisations introduced to speed up the simulation do not actually omit checking relevant behaviours of the system under verification.

The aim of this paper is to fill the above gap by presenting a formal semantics for simulators.

1 Introduction

System Level Verification (SLV) of Cyber-physical Systems has the goal of verifying that the *whole* (i.e., software + hardware) system meets the given specifications. Hardware In the Loop Simulation (HILS) is currently the main workhorse for system level verification and is supported by *Model Based Design* tools such as Simulink, (http://www.mathworks.com) VisSim, (http://www.vissim.com) and Modelica (https://www.modelica.org/). In HILS the *actual software* reads [sends] values from [to] mathematical models (*simulation*) of the physical systems (e.g. engines, analog circuits, etc.) it will be interacting with.

Our System Under Verification (SUV) state can take discrete values (e.g., from the software state) as well as continuous values (e.g., from the physical system state). Thus our SUV can be conveniently modelled as a *Hybrid System* (e.g., see [2] and citations thereof) whose inputs belong to a finite set of uncontrollable events (*disturbances*) modelling failures in sensors or actuators, variations in the system parameters, etc.

We focus on *deterministic systems* (the typical case for control systems), and model nondeterministic behaviours (such as faults) with disturbances. Accordingly, in our framework, a *simulation scenario* is just a finite sequence of disturbances.

A system is expected to *withstand* all disturbance sequences that may arise in its operational scenarios. Correctness of a system is thus defined with respect to such *admissible* disturbance sequences and the goal of HILS is exactly that of showing that indeed the considered SUV can withstand all admissible disturbance sequences. The set of admissible disturbance sequences typically satisfies constraints like the following: 1) the number of failures occurring within a certain period of time is less than a given threshold; 2) the time interval between two consecutive failures is greater than a given threshold; 3) a failure is repaired within a certain time, etc.

We focus on HILS based *Bounded* SLFV of *safety* properties. That is, given a time horizon T and a time step τ (time quantum between disturbances) our HILS campaign returns *PASS* if there is no *admissible* disturbance sequence of length T and time step τ that violates the property under verification

J. Esparza, E. Tronci (Eds.): Games, Automata,
Logics and Formal Verification 2015 (GandALF'15)
EPTCS 193, 2015, pp. 86–99, doi:10.4204/EPTCS.193.7

and *FAIL*, along with a counterexample, otherwise. In other words, *Bounded* SLFV is an *exhaustive* (with respect to the set of admissible disturbance sequences) HILS campaign. In such a framework our exhaustive HILS campaign works as a *black box bounded model checker* where the SUV behaviour is defined by a simulator.

1.1 Motivations

In our context the number of admissible disturbance sequences is finite since the number of disturbances is finite and the time horizon as well as the time quantum between disturbances are both bounded. Nevertheless the number of admissible disturbance sequences can be quite large. As a result, depending on the system considered and on the degree of assurance sought a HILS campaign may easily require months of simulation activity.

To decrease such a simulation time many HILS based *Bounded Model Checking* approaches to SLFV have been devised. Typically such approaches save simulation time by avoiding simulating more than once the same sequence of disturbances. This, in turn, is attained by exploiting the capability of simulators to save and restore a simulation state.

However, even though such approaches aim to (bounded) formal verification, as a matter of fact the simulator behaviour is never formally modelled and the proof of correctness of the proposed approaches basically relies on the intuitive notion of simulator behaviour. This gap makes it hard to check if the optimisations introduced to speed up the simulation do not actually omit checking of relevant behaviours of the system under verification.

The aim of this paper is to fill the above gap by presenting a *formal* semantics for simulators and by proving *soundness* and *completeness* properties for it.

1.2 Main Contributions

Our main contributions can be summarised as follows.

We give a formal notion of simulator, of simulation campaign, and provide a formal *operational semantics* for simulators.

We show *soundness* of our simulator semantics by showing that any simulation campaign defines a set of (*in silico*) experiments that can be carried out on our SUV.

We show *completeness* of our simulator semantics by showing that any set of (*in silico*) experiments to be carried out on our SUV can be defined with a simulation campaign.

1.3 Related Work

SLFV of cyber-physical systems via HILS based bounded model checking has been studied in many contexts. Here are a few examples. Formal verification of Simulink models has been investigated in [22, 18, 25] focusing on discrete time models (e.g., Stateflow or Simulink restricted to discrete time operators) with small domain variables. Formal verification of fully general Simulink models has been investigated in [12, 13, 14, 15]. Formal verification of satellite operational procedures using ESA SIMSAT simulator has been investigated in [7].

Simulation based approaches to statistical model checking have been widely investigated. Here are just a few examples: Simulink models for cyber-physical systems have been studied in [28], mixed-analog circuits have been analyzed in [5], smart grid control policies have been considered in [16], and biological models have been studied in [19, 23, 17].

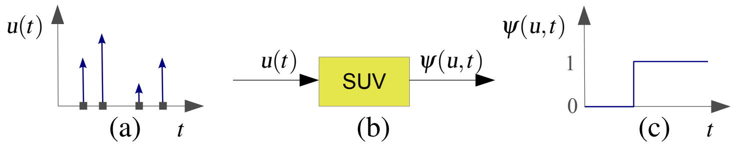


Figure 1: (a) a discrete event sequence $u \in \mathcal{U}^{\mathbb{R}^{\geq 0}}$; (b) our SUV; (c) the SUV output $\psi(u,t)$

Of course *Model Based Testing* (e.g., see [4]) has widely considered automatic generation of test cases from models. In our HILS setting, automatic generation of simulation scenarios (for Simulink) has been investigated, for example, in [8, 10, 3, 24].

Finally, synergies between simulation and formal methods have been widely investigated also in digital hardware verification. Examples are in [26, 9, 20, 6] and citations thereof.

All simulation based verification approaches considered in the literature heavily rely on carefully driving simulators in order to effectively carry out the planned verification activity. However, to the best of our knowledge, none of them addresses the issue of providing a simulator semantics accounting for the simulator commands enabling saving and restoring of simulation states (the main simulation commands used to save simulation time).

1.4 Outline of the paper

Section 2 describes how we model disturbances as uncontrollable inputs to our (cyber-physical) SUV that, in turn, is modelled as a *discrete event system*. Section 3 formalises the notion of simulator, simulation campaign and simulator semantics. Section 4 and Section 5 provide, respectively, soundness and completeness theorems for our simulator semantics.

2 Dynamical Systems

In this section we give the formal background on which our approach rests. To this end, we model disturbances (Definition 1) and define our system (Definition 4), by rephrasing the definition of dynamical system (see, e.g., [21]). Then, we define the simulation scenario, that is the sequence of disturbances occurring when the system starts from a given state, and the set of transitions associated to it.

Throughout the paper, we denote $\mathbb{N}$ the set of natural numbers, $\mathbb{N}^+$ the set of positive natural numbers, $\mathbb{R}^+$, $\mathbb{R}^{\geq 0}$ and $\mathbb{R}$ the sets of positive, non-negative and all real numbers, respectively. Throughout the paper, we use $\mathbb{R}^{\geq 0}$ to represent time and $\mathbb{R}^+$ to represent non-zero time durations.

A *discrete event sequence* (Definition 1 and Figure 1(a)), is a function associating to each (continuous) time instant a disturbance event (such as a fault, a variation in a system parameter, etc). We are considering a bounded time horizon, accordingly we require that the number of disturbances is finite, since no system can withstand an infinite number of disturbances within a finite time. We represent with the real number 0 the event carrying no disturbance and with nonzero reals actual disturbances.

Definition 1 (Discrete event sequence) *Let $\mathcal{U}$ be a finite subset of $\mathbb{R}$ such that $0 \in \mathcal{U}$. A* discrete event sequence *over $\mathcal{U}$ is a function $u : \mathbb{R}^{\geq 0} \to \mathcal{U}$ such that the set $\{t \in \mathbb{R}^{\geq 0} \mid u(t) \neq 0\}$ has finite cardinality. We denote with $\mathcal{U}^{\mathbb{R}^{\geq 0}}$ the set of discrete event sequences over $\mathcal{U}$. We call* time horizon *of a discrete event sequence u the value* $\max\{t \in \mathbb{R}^{\geq 0} \mid u(t) \neq 0\}$.

An *event list* provides an explicit representation for a discrete event sequence by listing pairs (τ, e), where $e > 0$ is an event and τ is the time elapsed since the last (nonzero) event.

Example 1 (Discrete event sequence and event list) *As a first example, let us consider the function u defined as follows:*

$$u(t) = \begin{cases} 1 & \textit{for} \quad t = k\tau, \textit{ where } k = 1,2,3 \textit{ and } \tau = 1 \\ 0 & \textit{otherwise} \end{cases}$$

The corresponding event list is: $[(\tau,1),(\tau,1),(\tau,1)]$*, and the time horizon of u is* 3τ*.*

Remark 1 *Let* $\delta(t)$*, with* $t \in \mathbb{R}$*, be the function such that if* $t = 0$ *then* $\delta(t) = 1$*, else* $\delta(t) = 0$*, that is* $\delta(t)$ *represents a discrete impulse. Any discrete event sequence* $u(t)$ *with horizon h can be written in a unique way as finite sum of discrete impulses, that is:*

$$u(t) = p_0\delta(t) + \sum_{i=1}^{h} p_i \delta(t - \sum_{k=0}^{i-1} \tau_k)$$

where $p_0 \in \mathscr{U}$*,* $p_i \in (\mathscr{U} - \{0\})$*, and* $\tau_k \in \mathbb{R}^+$*.*

Example 2 (Impulses) *Let us consider the discrete event sequence u represented in Figure 1(a). It can be defined using impulses as follows:*

$$u(t) = 2\delta(t-3) + 3\delta(t-5) + \delta(t-10) + 2\delta(t-13)$$

The event list associated to u is: $[(3\tau,2),(2\tau,3),(5\tau,1),\ (3\tau,2)]$.

Definition 4 formalizes how we model our SUV, whereas Definition 2 and 3 define, respectively, the subset of $\mathscr{U}^{\mathbb{R}^{\geq 0}}$ obtained as a restriction to a real interval, and the concatenation of two such subsets.

Definition 2 *Let* $\mathscr{U}^{\mathbb{R}^{\geq 0}}$ *be the set of discrete event sequences over the set* $\mathscr{U}$*. Given a discrete event sequence* $u \in \mathscr{U}^{\mathbb{R}^{\geq 0}}$ *and two positive real numbers* $t_1 \leq t_2$*, we denote with* $u\mid_{[t_1,t_2)}$ *the restriction of u to the interval* $[t_1,t_2)$*, i.e. the function* $u\mid_{[t_1,t_2)}: [t_1,t_2) \to \mathscr{U}$*, such that* $u\mid_{[t_1,t_2)}(t) = u(t)$ *for all* $t \in [t_1,t_2)$*. We denote* $\mathscr{U}^{[t_1,t_2)}$ *the* restriction *of* $\mathscr{U}^{\mathbb{R}^{\geq 0}}$ *to the domain* $[t_1,t_2)$*.*

Definition 3 *Assume that* $t_1,t_2,t_3 \in \mathbb{R}^{\geq 0}$ *such that* $t_1 < t_2 < t_3$*. If* $\omega \in \mathscr{U}^{[t_1,t_2)}$ *and* $\omega' \in \mathscr{U}^{[t_2,t_3)}$*, their* concatenation*, denoted as* $\omega\omega'$*, is the function* $\tilde{\omega} \in \mathscr{U}^{[t_1,t_3)}$ *defined as:*

$$\tilde{\omega}(t) = \begin{cases} \omega(t) & \textit{if } t \in [t_1,t_2) \\ \omega'(t) & \textit{if } t \in [t_2,t_3) \end{cases}$$

In our setting the system to be verified can be modelled as a continuous time *Input-State-Output* deterministic dynamical system (see e.g. [21]) whose input functions are discrete event sequences, whose state can undertake continuous as well as discrete changes, and whose output ranges on any combination of discrete and continuous values.

Definition 4 (Discrete Event System) *A* Discrete Event System*, or simply DES,* $\mathscr{H}$ *is a tuple* $(\mathscr{X},\mathscr{U},\mathscr{Y},\varphi,\psi)$*, where:*

- $\mathscr{X}$*, the* state space *of* $\mathscr{H}$*, is a non-empty set whose elements denote* states*;*
- $\mathscr{U}$*, the* input value space *of* $\mathscr{H}$*, is a finite subset of* $\mathbb{R}$ *such that* $0 \in \mathscr{U}$*;*

- $\mathscr{Y}$*, the* output value space *of* $\mathscr{H}$*, is a non-empty set of* outputs*;*
- $\varphi : \mathbb{R}^{+} \times \mathscr{X} \times \mathscr{U}^{\mathbb{R}^{\geq 0}} \to \mathscr{X}$ *is the* transition map *of* $\mathscr{H}$. *Function* φ *must satisfy the following properties:*
  - semigroup*: for each* $t_1, t_2, t_3 \in \mathbb{R}^{\geq 0}$ *such that* $t_1 < t_2 < t_3$, $\omega \in \mathscr{U}^{[t_1,t_2)}$, $\omega' \in \mathscr{U}^{[t_2,t_3)}$, $x \in \mathscr{X}$ *we have that* $\omega\omega' \in \mathscr{U}^{[t_1,t_3)}$ *is such that* $\varphi(t_3 - t_1, x, \omega\omega') = \varphi(t_3 - t_2, \varphi(t_2 - t_1, x, \omega), \omega')$*;*
  - consistency*: for each* $u \in \mathscr{U}$, $x \in \mathscr{X}$*, we have* $\varphi(0, x, u) = x$*;*
- $\psi : \mathbb{R}^{\geq 0} \times \mathscr{X} \to \mathscr{Y}$ *is the* observation function *of* $\mathscr{H}$.

Note that any simulator driven by its script language can be seen as a discrete event system. This is why we focus on DES.

Our approach can model both the case in which the input is controllable, for example by control software (Example 3), and the case in which the input is uncontrollable, for example disturbances such as faults (Examples 4 and 5).

Example 3 (Inverted Pendulum) *A simple system is given by the Inverted Pendulum with Stationary Pivot Point, see e.g. [11, 1]. The system is modeled by taking the angle* θ *and the angular velocity* $\dot{\theta}$ *as state variables. The input of the system is the torquing force u, that can influence the velocity in both directions. Moreover, the behaviour of the system depends on the pendulum mass m, the length of the pendulum l and the gravitational acceleration g. Given such parameters, the motion of the system is described by the differential equation* $\ddot{\theta} = \frac{g}{l} sin\theta + \frac{1}{ml^2}u$.

Let $\mathscr{U}$ *be* $\{-1, 0, 1\}$, $\tau = 10^{-6}$. *Our discrete event system* $\mathscr{H}$ *is the tuple* $(\mathscr{X}, \mathscr{U}, \mathscr{Y}, \varphi, \psi)$*, where:*

- $\mathscr{X} = \mathbb{R}^2$ *and* $\mathscr{Y} = \mathbb{R}^2$*;*
- φ *is solution to the system of differential equations:*

  $\dot{x_1} = x_2$

  $\dot{x_2} = \frac{g}{l} sin x_1 + \frac{1}{ml^2} u$

  where x_1 *is the angle* θ *and* x_2 *is the angular velocity* $\dot{\theta}$*;*
- $\psi(t)$ *is given by* $[x_1(t), x_2(t)]$.

In Figure 2 the Simulink model of the inverted pendulum is shown, where we assume the pendulum mass $m = 1$ *and the length of the pendulum* $l = 1$. *Also we assume the function u is given to the model as a sequence of values in the set* $\{-1, 0, 1\}$.

Example 4 (Inverted pendulum on cart) *Another example is given by the inverted pendulum on cart. For this system, the control input is the force F that moves the cart horizontally and the outputs are the angular position of the pendulum* θ *and the horizontal position of the cart x. The physical constraint between the cart and pendulum gives that both the cart and the pendulum have one degree of freedom each (x and* θ*, respectively). The controlled system (the* plant*) consists of the cart and the pendulum, whereas the controller consists of the control software computing F from the plant outputs (x and* θ*). The dynamics of the system is described in the example available in the Simulink distribution. The Simulink model of the pendulum on cart, where disturbances are added, is shown in Figure 3.*

The system state is a pair (z, w) *where z is the state of the control software and w is the plant state, namely* $w = [w_1, w_2, w_3, w_4]$*, where:* w_1 *= cart position,* w_2 *=cart velocity ,* w_3 *= pendulum angle,* w_4 *= pendulum angular velocity.*

We model irregularities in the cart rail with a disturbance on the cart weight with respect to its nominal value 0.455 kg. Let $\mathscr{U} = \{0, 1, 2\}$ *be our set of disturbances (see Definition 1) modelling the fact that the cart weight is* $(d + 0.455)$*, with* $d \in \mathscr{U}$.

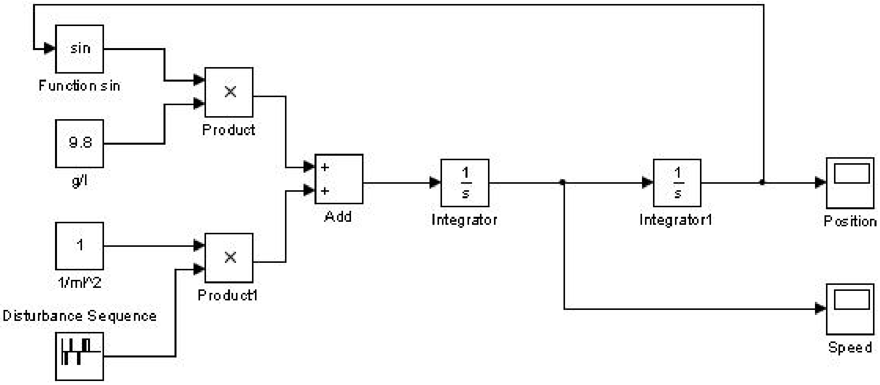


Figure 2: Simulink model of the inverted pendulum.

We will use the Inverted pendulum on cart (Example 4) as running example throughout the paper.

Example 5 (Fuel Control System) *The Fuel Control System (FCS) model in the Simulink distribution (see Figure 4) has been studied in [27] using statistical model checking techniques, whereas the formal verification has been discussed in [12, 14]. The model is equipped with four sensors: throttle angle, speed, Oxygen in Exhaust Gas (EGO) and Manifold Absolute Pressure (MAP). In this case, the tuple* $(\mathscr{X}, \mathscr{U}, \mathscr{Y}, \varphi, \psi)$ *representing the discrete event system is:*

- $\mathscr{X}$ *is the set of plant (i.e., the engine) states along with the control software states;*
- $\mathscr{Y}$ *is the set of plant outputs monitored by the control software;*
- $\mathscr{U}$ *is the set of disturbance sequences that can be obtained assuming that only sensors EGO and MAP can fail, giving rise to disturbances 1 and 2, respectively; the minimum time between faults is one second and all faults are transient, that is disturbance 1 models a fault on sensor EGO, followed by a repair within one second, and disturbance 2 models a fault on sensor MAP, followed by a repair within one second too;*
- φ *computes the dynamics of the system states;*
- $\psi(t)$ *computes the system output from the present system state.*

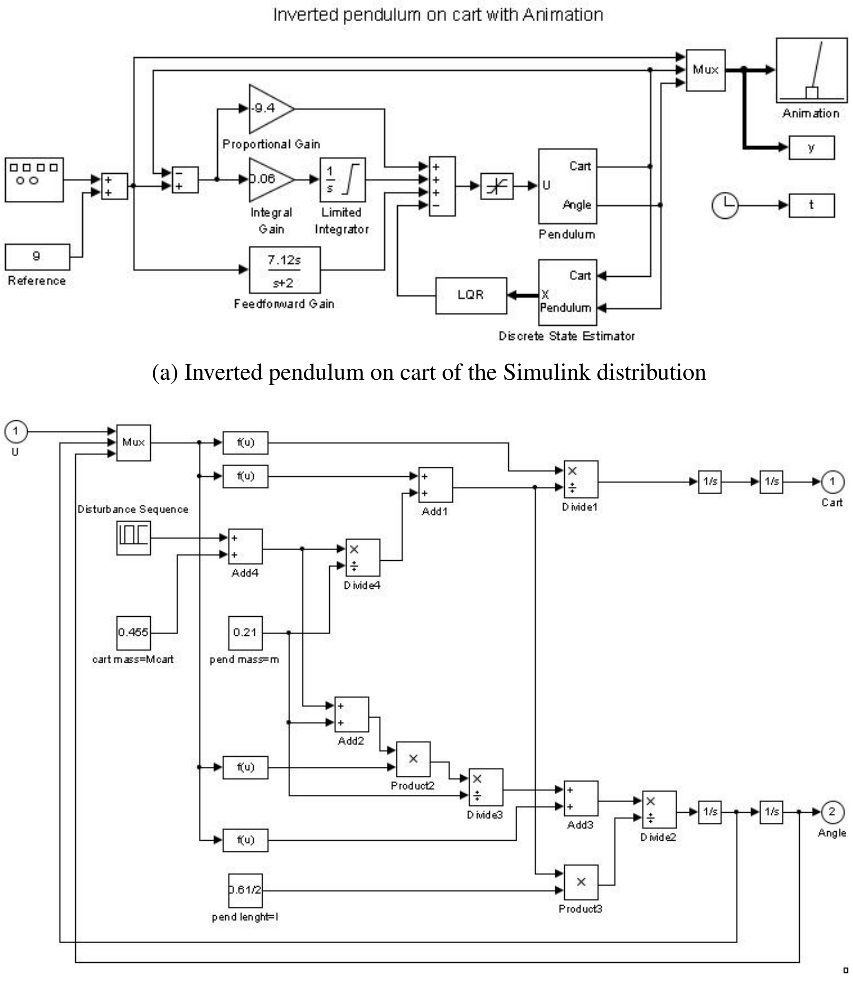
Our discrete event system as defined in Definition 4, models a hybrid system describing a cyber physical system, as shown in Examples 3, 4 and 5. For this reason, we denote our system with $\mathscr{H}$.

In the following we define the notion of simulation scenario, that is the sequence of disturbances received by our system starting from a given initial state, and we give an example.

Definition 5 (Simulation scenario) *A* simulation scenario *for* $\mathscr{H}$ *is a pair* (x, u) *where* $x \in \mathscr{X}$ *and* $u \in \mathscr{U}^{\mathbb{R}^{\geq 0}}$.

Example 6 (Simulation scenario) *Let* $\mathscr{H}$ *be the Inverted pendulum on cart system described in Example 4. Let* $u(t)$ *be the discrete event sequence defined as:* $u(t) = \delta(t - 0.04) + \delta(t - 0.08)$ *and let the initial state be* $x_0 = (z_0, [0,0,0,0])$, *where* z_0 *is the control software initial state. Then, a simulation scenario for* $\mathscr{H}$ *is* (x_0, u).

Definitions 6 and 7 give the definition of sequence of transitions and set of transitions explored by a SUV under a given simulation scenario, respectively.



(a) Inverted pendulum on cart of the Simulink distribution

(b) Cart model with disturbances

Figure 3: Simulink model of the pendulum on cart with disturbances.

Definition 6 (Trace of a simulation scenario) *Let $\mathscr{H}$ be a DES and let $x \in \mathscr{X}$ be a state and $u \in \mathscr{U}^{\mathbb{R}^{\geq 0}}$, $u(t) = p_0\delta(t) + \sum_{i=1}^{h} p_i\delta(t - \sum_{k=0}^{i-1}\tau_k)$, a discrete event sequence giving a simulation scenario (x,u) for $\mathscr{H}$. The* trace *of the simulation scenario (x,u), denoted $Tr(x,u)$, is the finite sequence of transitions $Tr(x,u) = [(x_0, p_0, 0, x_1), (x_1, p_1, \tau_1, x_2), \ldots, (x_{h-1}, p_{h-1}, \tau_{h-1}, x_h)]$ such that $x_0 = x$ and $x_{i+1} = \varphi(\tau_i, x_i, p_i\delta(t))$.*

Example 7 (Trace of a simulation scenario) *Let $\mathscr{H}$ be the Inverted pendulum on cart system described in Example 4 and let (x_0, u) be the simulation scenario of Example 6.*

The trace of (x_0,u) is $Tr(x_0,u) = [(x_0,0,0,x_1),(x_1,1,0.04,x_2),(x_2,1,0.04,x_3)]$, where $x_0 = (z,[0,0,0,0])$ and the x_i values, $i = 1,2,3$ are obtained by running the simulation with the Simulink model shown in Example 4 and are: $x_1 = (z_1,[-0.017,-0.881,0.057,2.914])$, $x_2 = (z_2,[-0.049,-0.694,0.167,2.451])$ and $x_3 = (z_3,[-0.072,-0,431,0.253,1.878])$.

Definition 7 (Set of transitions of a simulation scenario) *The set of transitions associated to a simulation scenario (x,u) is the set:*

$$\mathscr{T}_{(x,u)} = \{(z,p,\tau,z') | (z,p,\tau,z') \in Tr(x,u)\}$$

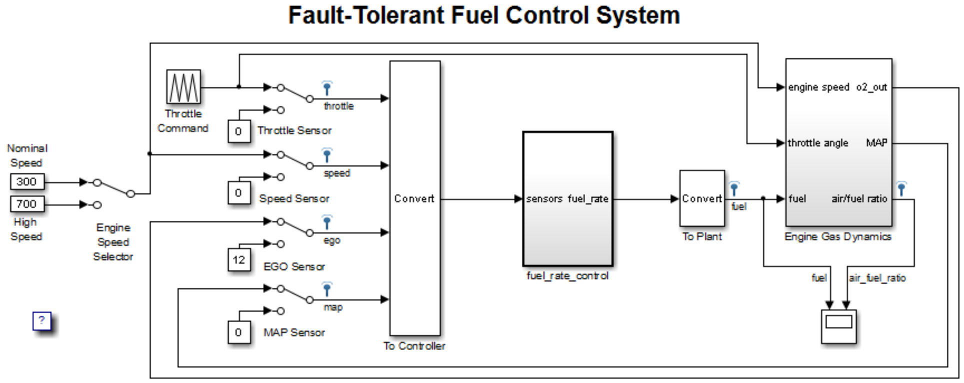


Figure 4: The Simulink Fuel Control System.

Example 8 (Set of transitions of a simulation scenario) *Let us consider system $\mathscr{H}$, simulation scenario (x_0, u), and trace $Tr(x_0, u)$ as in Example 6. The set of transitions associated to (x_0, u) is simply the set $\mathscr{T}_{(x,u)} = \{(x_0, 0, 0, x_1), (x_1, 1, 0.04, x_2), (x_2, 1, 0.04, x_3)\}$, where x_1, x_2 and x_3 assume the values specified in Example 6.*

3 Simulators and Simulation Campaigns

In this Section we formalise the notion of discrete event system simulator (Definition 8 and Definition 9), of simulation campaign (Definition 10) and of set of transitions of a simulation campaign (Definition 12).

In many cases it is necessary to consider a huge number of simulation scenarios for having an exhaustive HILS. The overall number of simulation steps can be prohibitively large if each scenario is simulated from the initial state of the (SUV) simulator. The definition of set of transitions of a simulation campaign (Definition 12) helps to individuate states necessary to complete the simulation campaign avoiding to repeat the same sequence of commands several times.

Definition 8 (Discrete Event System Simulator) *A Discrete Event System (DES) simulator $\mathscr{S}$ is a tuple $(\mathscr{H}, W)$, where $\mathscr{H} = (\mathscr{X}, \mathscr{U}, \mathscr{Y}, \varphi, \psi)$ is a DES and W is a finite set whose elements are called* simulator states. *Each $w \in W$ is a pair (x, M) where $x \in \mathscr{X}$ is a state of $\mathscr{H}$ and M is a finite subset of $\mathscr{X}$ that models the content of the* simulator memory.

Unless otherwise stated, in the following $\mathscr{S}$ is a simulator for the DES $\mathscr{H}$ as in Definition 8.

Note that, at the beginning the simulator memory contains the initial state x_0 of $\mathscr{H}$.

The semantics of simulator commands we use to execute our simulation scenarios, and the transition function ξ are given in Definition 9.

Definition 9 (Simulator commands and transition function) *Let $\mathscr{S}$ be a simulator.*

- *The commands for $\mathscr{S}$ are:* load(x), store, free(x), run(p, t), *where $x \in \mathscr{X}$ is a state of $\mathscr{H}$, $t \in \mathbb{R}^+$ is a time duration, and $p \in \mathscr{U}$ is an event (x, t, p are command arguments).*

- *The* transition function ξ *of* $\mathscr{S}$*, defines how the internal state of the simulator* $\mathscr{S}$ *changes upon execution of a command. Namely:* $\xi(x, M, \mathrm{cmd}(\mathrm{args})) = (x', M')$ *when the simulator* $\mathscr{S}$ *moves from internal state* (x, M) *to state* (x', M') *upon processing command* cmd *with arguments* args*.*

  For each $x \in \mathscr{X}$*, function* ξ *is defined as follows:*

  - *if* $x' \in M$ *then* $\xi(x, M, \mathrm{load}(x')) = (x', M)$
  - *if* $x' \in M$ *then* $\xi(x, M, \mathrm{free}(x')) = (x, M \setminus \{x'\})$
  - $\xi(x, M, \mathrm{store}) = (x, M \cup \{x\})$
  - $\xi(x, M, \mathrm{run}(p, \tau)) = (x', M)$*, where* $x' = \varphi(\tau, x, u)$*, where* $u(t) = p\delta(t)$*.*

Given a sequence of simulation scenarios, we can build a sequence of commands, *simulation campaign*, driving the simulator through such scenarios. We define the simulator *output sequence* as the sequence of the SUV outputs associated to the simulator states traversed by a simulation campaign. Conversely, given a simulation campaign, we can compute the sequence of scenarios simulated by it. These concepts are formalised in Definition 10.

Definition 10 (Simulation campaign and sequence of simulator states) *Let* $\mathscr{S}$ *be a simulator and let* ξ *be its transition function.*

- *A* simulation campaign *for* $\mathscr{S}$ *is a triple* $\Xi = (x, M, \chi)$ *, where* $x \in \mathscr{X}$*,* $M \subset \mathscr{X}$ *and* χ *is a sequence (possibly empty or infinite) of commands along with their arguments,* $\chi = \mathrm{cmd}_0(\mathrm{args}_0), \mathrm{cmd}_1(\mathrm{args}_1), \ldots$*. A simulation campaign consisting of a finite sequence of commands is a* finite *simulation campaign or a simulation campaign of finite length; the length of* $\chi = \mathrm{cmd}_0(\mathrm{args}_0), \ldots, \mathrm{cmd}_{c-1}(\mathrm{args}_{c-1})$ *is c and it is denoted by* $|\chi| = c$*.*
- *The* sequence of simulator states *of* $\mathscr{S}$ *with respect to a simulation campaign* $\Xi = (x_0, M_0, \chi)$ *is the sequence* $(x_0, M_0), (x_1, M_1), \ldots$*, where for all j,* $\xi(x_j, M_j, \mathrm{cmd}_j(\mathrm{args}_j)) = (x_{j+1}, M_{j+1})$*.*

  We denote with $\chi(x_0, M_0, j)$ *the j-th element of such a sequence, that is* $\chi(x_0, M_0, j) = (x_j, M_j)$*. In other words* $\chi(x_0, M_0, j)$ *is the simulator state after the execution of the j-th command.*
- *The set of simulator states with respect to a simulation campaign* χ *is denoted* $\chi(x_0, M_0)$*, that is* $\chi(x_0, M_0) = \{\chi(x_0, M_0, j) | j = 0, 1, \ldots, |\chi| - 1\}$*.*

Example 9 (Simulation campaign) *Let* $\mathscr{H}$ *be the Inverted pendulum on cart considered in Example 4 and let* (x_0, u) *be the simulation scenario considered in Example 6, where* $u(t) = \delta(t - 0.04) + \delta(t - 0.08)$*.*

The simulation campaign Ξ *obtained by using this simulation scenario is the triple* $\Xi = (x_0, \{x_0\}, \chi)$*, where* χ *is the sequence of commands* $\chi = (\mathrm{run}(0, 0.04), \mathrm{run}(1, 0.04), \mathrm{run}(1, 0.04))$*.*

The sequence of simulator states with respect to Ξ *is:*

$$(x_0, \{x_0\}) \xrightarrow{run(0,0.04)} (x_1, \{x_0\}) \xrightarrow{run(1,0.04)} (x_2, \{x_0\}) \xrightarrow{run(0,0.04)} (x_3, \{x_0\}).$$

State values are obtained by running the simulation.

An example of a more complex simulation campaign, Ξ_1*, can be obtained by considering the sequence of simulation scenarios* $((x_0, u), (x_3, u_1), (x_3, u_2), (x_0, u_3))$*, where:*
$u(t) = \delta(t - 0.04) + \delta(t - 0.08)$
$u_1(t) = \delta(t - 0.04) + \delta(t - 0.08) + \delta(t - 0.12)$
$u_2(t) = \delta(t - 0.12)$
$u_3(t) = \delta(t - 0.04) + 2\delta(t - 0.12) + \delta(t - 0.16) + 2\delta(t - 0.24)$.

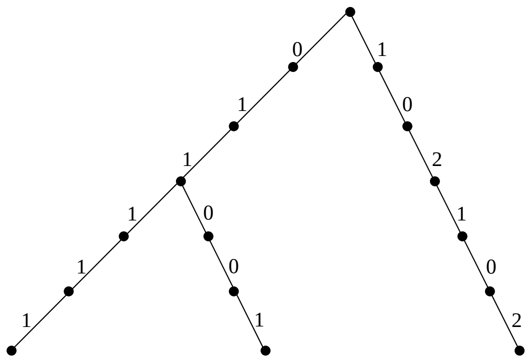


Figure 5: A graphical representation of simulation campaign Ξ_1 (Example 9).

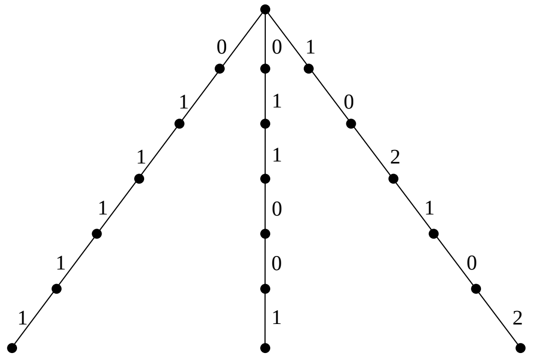


Figure 6: A graphical representation of the normal simulation campaign Ξ_2 (Example 10).

A graphical representation of simulation campaign Ξ_1 is shown in Figure 5, where we see that the discrete event sequences $u(t)$ and $u_3(t)$ are applied when having state x_0, and sequences u_1 and u_2 are applied having state x_3.

The simulation campaign Ξ_1 obtained by using the sequence of simulation scenarios above is the triple $\Xi_1 = (x_0, \{x_0\}, \chi_1)$, where χ_1 is given by the following command sequence:

$$\begin{aligned}\chi_1 = \quad & \mathit{run}(0,0.04),\ \mathit{run}(1,0.04),\ \mathit{run}(1,0.04),\ \mathit{store},\\ & \mathit{run}(1,0.04),\ \mathit{run}(1,0.04),\ \mathit{run}(1,0.04),\ \mathit{load}(x_3),\\ & \mathit{run}(0,0.04),\ \mathit{run}(0,0.04),\ \mathit{run}(1,0.04),\ \mathit{free}(x_3),\\ & \mathit{load}(x_0),\ \mathit{run}(1,0.04),\ \mathit{run}(0,0.04),\ \mathit{run}(2,0.04),\\ & \mathit{run}(1,0.04),\ \mathit{run}(0,0.04),\ \mathit{run}(2,0.04)\end{aligned}$$

The sequence of simulator states with respect to Ξ_1 is:

$$(x_0, \{x_0\}) \xrightarrow{\mathit{run}(0,0.04)} (x_1, \{x_0\}) \xrightarrow{\mathit{run}(1,0.04)} (x_2, \{x_0\}) \xrightarrow{\mathit{run}(1,0.04)} (x_3, \{x_0\}) \xrightarrow{\mathit{store}}$$
$$(x_3, \{x_0, x_3\}) \xrightarrow{\mathit{run}(1,0.04)} (x_4, \{x_0, x_3\}) \xrightarrow{\mathit{run}(1,0.04)} (x_5, \{x_0, x_3\}) \xrightarrow{\mathit{run}(1,0.04)} (x_6, \{x_0, x_3\}) \xrightarrow{\mathit{load}(x_3)}$$
$$(x_3, \{x_0, x_3\}) \xrightarrow{\mathit{run}(0,0.04)} (x_7, \{x_0, x_3\}) \xrightarrow{\mathit{run}(0,0.04)} (x_8, \{x_0, x_3\}) \xrightarrow{\mathit{run}(1,0.04)} (x_9, \{x_0, x_3\}) \xrightarrow{\mathit{free}(x_3)}$$
$$(x_9, \{x_0\}) \xrightarrow{\mathit{load}(x_0)} (x_0, \{x_0\}) \xrightarrow{\mathit{run}(1,0.04)} (x_{10}, \{x_0\}) \xrightarrow{\mathit{run}(0,0.04)} (x_{11}, \{x_0\}) \xrightarrow{\mathit{run}(2,0.04)} (x_{12}, \{x_0\}) \xrightarrow{\mathit{run}(1,0.04)}$$
$$(x_{13}, \{x_0\}) \xrightarrow{\mathit{run}(0,0.04)} (x_{14}, \{x_0\}) \xrightarrow{\mathit{run}(2,0.04)} (x_{15}, \{x_0\})$$

Definition 11 gives the notion of normal simulation campaign, that is a simulation campaign for which every simulation scenario starts from an initial state.

Definition 11 (Normal Simulation Campaign) *A simulation campaign is in* normal *form if it consists only of commands* load *and* run*.*

Example 10 (Normal Simulation Campaign) *An example of simulation campaign in normal form is $\Xi_2 = (x_0, \{x_0\}, \chi_2)$, where χ_2 is given by the following command sequence:*

$$\begin{aligned}\chi_2 = \quad & \mathit{run}(0,0.04),\ \mathit{run}(1,0.04),\ \mathit{run}(1,0.04),\ \mathit{run}(1,0.04),\ \mathit{run}(1,0.04),\ \mathit{run}(1,0.04),\\ & \mathit{load}(x_0),\ \mathit{run}(0,0.04),\ \mathit{run}(1,0.04),\ \mathit{run}(1,0.04),\ \mathit{run}(0,0.04),\ \mathit{run}(0,0.04),\ \mathit{run}(1,0.04),\\ & \mathit{load}(x_0),\ \mathit{run}(1,0.04),\ \mathit{run}(0,0.04),\ \mathit{run}(2,0.04),\ \mathit{run}(1,0.04),\ \mathit{run}(0,0.04),\ \mathit{run}(2,0.04)\end{aligned}$$

A graphical representation of Ξ_2 is shown in Figure 6.

Note that, since a normal simulation campaign has no *store* commands, a command *load* can only load an initial state.

Definition 12, resting on Definition 10, defines the set of transitions of a simulation campaign. Definition 13 gives the notion of equivalent simulation campaigns.

Definition 12 (Set of transitions of a Simulation Campaign) *We denote with $\mathscr{T}_\Xi$ the set of transitions of $\mathscr{S}$ explored by Ξ, that is $\mathscr{T}_\Xi = \{(x,p,\tau,x')|\exists\, M,M'\ [(x,M)$ is a simulator state of $\mathscr{S}$ wrt $\Xi\ \wedge\ \xi(x,M,run(p,\tau)) = (x',M')]\}$.*

Definition 13 (Equivalent simulation campaigns) *We say that the simulation campaign Ξ is equivalent to Ξ' and we write $\Xi \sim \Xi'$ if $\mathscr{T}_\Xi = \mathscr{T}_{\Xi'}$.*

Example 11 (Equivalent simulation campaigns) *The simulation campaign Ξ_1 in Example 9 and the normal simulation campaign Ξ_2 in Example 10 are equivalent.*

In fact the set of transitions explored by Ξ_1 and Ξ_2 is:

$\mathscr{T}_{\Xi_1} = \mathscr{T}_{\Xi_2} = \{\ (x_0,0,0.04,x_1),\ (x_1,1,0.04,x_2),\ (x_2,1,0.04,x_3),\ (x_3,1,0.04,x_4),\ (x_4,1,0.04,x_5),$ $(x_5,1,0.04,x_6),\ (x_3,0,0.04,x_7),\ (x_7,0,0.04,x_8),\ (x_8,1,0.04,x_9),\ (x_0,1,0.04,x_{10}),\ (x_{10},0,0.04,x_{11}),$ $(x_{11},2,0.04,x_{12}),\ (x_{12},1,0.04,x_{13}),\ (x_{13},0,0.04,x_{14}),\ (x_{14},2,0.04,x_{15})\}$

This can also be easily seen looking at the set of edges (transitions) in Figures 5 and 6.

Lemma 1 formalizes the fact that for each simulation campaign, we can determine an equivalent simulation campaign in which each simulation scenario starts from an initial state.

Lemma 1 *Given a simulation campaign Ξ for a simulator $\mathscr{S}$, there exists a simulation campaign Ξ' such that:*

- *Ξ' is in normal form*
- *$\Xi' \sim \Xi$*

We give the idea of the proof by using the following example.

Example 12 (Lemma 1) *Consider the simulation campaign Ξ_1 in Example 9. Ξ_1 is not in normal form. However, by modifying it so that all simulation scenarios (paths on the tree of Figure 5) start from the initial state, we get the normal simulation campaign Ξ_2 illustrated in Example 10.*

Further, it follows from Example 10 that $\Xi_1 \sim \Xi_2$.

4 Soundness

In this section we show the soundness of our simulator semantics. That is, we show that any simulation campaign stems from a set of simulation scenarios. This guarantees that any simulation campaign has indeed a physical (computational) meaning.

Theorem 1 (Soundness) *Given a simulation campaign Ξ for $\mathscr{S}$ there exists a set $\mathscr{A} = \{(x_1,u_1),(x_2,u_2),\ldots,(x_k,u_k)\}$ of simulation scenarios such that*

$$\mathscr{T}_\Xi = \cup\{\mathscr{T}_{(x_i,u_i)}|\ i = 1,\ldots,k\}.$$

As we did for Lemma 1, we give the idea of the proof by using an example.

Example 13 (Soundness) *Consider the simulation campaign Ξ_1 in Example 9. The set of transitions of Ξ_1, $\mathscr{T}_{\Xi_1}$, is shown in Example 11.*

Now, let us consider the set $\mathscr{A}$ consisting of simulation scenarios (x_0,u), (x_3,u_1), (x_3,u_2) and (x_0,u_3), defined in Example 9.

The sets of transitions for simulation scenario (x_0,u) is shown in Example 8, and the sets of transitions associated to the other three simulation scenarios are, respectively:

$\mathscr{T}_1$ = $\mathscr{T}_{(x_3,u_1)} = \{\ (x_3,1,0.04,x_4),\ (x_4,1,0.04,x_5),\ (x_5,1,0.04,x_6)\}$

$\mathscr{T}_2$ = $\mathscr{T}_{(x_3,u_2)} = \{\ (x_3,0,0.04,x_7),\ (x_7,0,0.04,x_8),\ (x_8,1,0.04,x_9)\}$

$\mathscr{T}_3$ = $\mathscr{T}_{(x_0,u_3)} = \{\ (x_0,1,0.04,x_{10}),\ (x_{10},0,0.04,x_{11}),\ (x_{11},2,0.04,x_{12}),\ (x_{12},1,0.04,x_{13}),\ (x_{13},0,0.04,x_{14}),\ (x_{14},2,0.04,x_{15})\}$

It is easy to see that $\mathscr{T}_{\Xi_1} = \cup\{\mathscr{T}_j \mid j = 0,\ldots,3\}$.

5 Completeness

In this section we show the completeness of our simulator semantics. That is, we show that any set of simulation scenarios yields a simulation campaign. This guarantees that any set of physical experiments can be defined by a suitable simulation campaign.

Theorem 2 *Let $\mathscr{A} = \{(x_1,u_1),(x_2,u_2),\ldots,(x_k,u_k)\}$ be a set of simulation scenarios of $\mathscr{H}$. Then there exists a simulation campaign Ξ for $\mathscr{S}$ such that*

$$\mathscr{T}_\Xi = \cup\{\mathscr{T}_{(x_i,u_i)} \mid\ i = 1,\ldots,k\}.$$

Also for this theorem, we give the idea of the proof by using an example.

Example 14 (Completeness) *Let us consider the set $\mathscr{A}$ consisting of simulation scenarios (x_0,u), (x_3,u_1), (x_3,u_2) and (x_0,u_3), defined in Example 9.*

The sets of transitions associated to these simulation scenarios, $\mathscr{T}_0 = \mathscr{T}_{(x_0,u)}$, $\mathscr{T}_1 = \mathscr{T}_{(x_3,u_1)}$, $\mathscr{T}_2 = \mathscr{T}_{(x_3,u_2)}$ and $\mathscr{T}_3 = \mathscr{T}_{(x_0,u_3)}$, are shown in Example 13. Let $\bar{\mathscr{T}}$ be the set obtained as union of the sets of transitions above, that is $\bar{\mathscr{T}} = \cup\{\mathscr{T}_j \mid j = 0,\ldots,3\}$.

Now, let us consider the simulation campaign Ξ_1 in Example 9 and the set of transitions of Ξ_1, $\mathscr{T}_{\Xi_1}$, shown in Example 11.

It is easy to see that $\bar{\mathscr{T}} = \mathscr{T}_{\Xi_1}$.

6 Conclusions

We provided a formal notion of simulator, of simulation campaign. and a formal *operational semantics* for simulators.

Furthermore we showed *soundness* and *completeness* of our simulator semantics by showing that *any* simulation campaign defines a set of (*in silico*) experiments for the SUV (soundness) and, conversely, that *any* such a set can be defined with a simulation campaign (completeness).

This work enables formal proofs of correctness for simulation based formal verification approaches and provides formal tools enabling investigation of more aggressive approaches to the optimisation of the simulation activity entailed by HILS based SLFV.

Acknowledgements. Work partially supported by FP7 projects SmartHG (317761) and PAEON (600773).

References

[1] V. Alimguzhin, F. Mari, I. Melatti, I. Salvo & E.Tronci (2012): *Automatic control software synthesis for quantized discrete time hybrid systems*. In: *Proc. 51th IEEE Conference on Decision and Control, CDC*, doi:10.1109/CDC.2012.6426260.

[2] R. Alur (2011): *Formal verification of hybrid systems*. In: *Proc. 11th Int. Conf. on Embedded Software, EMSOFT 2011, part of the Seventh Embedded Systems Week*, ACM, doi:10.1145/2038642.2038685.

[3] A. Brillout, N. He, M. Mazzucchi, M. Purandare D. Kroening, P. Rümmer & G. Weissenbacher (2010): *Mutation-based Test Case Generation for Simulink Models*. In: *Proc. 8th Int. Conf. on Formal Methods for Components and Objects*, FMCO'09, Springer-Verlag, doi:10.1007/978-3-642-17071-3.

[4] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker & A. Pretschner (2005): *Model-Based Testing of Reactive Systems: Advanced Lectures*. *LNCS* 3472, Springer, doi:10.1007/b137241.

[5] E. M. Clarke, A. Donzé & A. Legay (2010): *On simulation-based probabilistic model checking of mixed-analog circuits*. *Formal Methods in System Design* 36(2), doi:10.1007/s10703-009-0076-y.

[6] F. M. De Paula & A. J. Hu (2007): *An effective guidance strategy for abstraction-guided simulation*. In: *Proc. 44th annual Design Automation Conference*, DAC '07, ACM, New York, NY, USA, doi:10.1145/1278480.1278498.

[7] Verzino G., F. Cavaliere, F. Mari, I. Melatti, G. Minei, I. Salvo, Y. Yushtein & E. Tronci (2012): *Model checking driven simulation of sat procedures*. In: *Proc. of 12th International Conference on Space Operations (SpaceOps 2012)*, SpaceOps, doi:10.2514/6.2012-1275611.

[8] A. A. Gadkari, A. Yeolekar, J. Suresh, S. Ramesh, S. Mohalik & K. C. Shashidhar (2008): *AutoMOTGen: Automatic Model Oriented Test Generator for Embedded Control Systems*. In: *Proc. 20th Int. Conf. Computer Aided Verification, CAV*, doi:10.1007/978-3-540-70545-1_19.

[9] P. H. Ho, T. Shiple, K. Harer, J. Kukula, R. Damiano, V. Bertacco, J. Taylor & J. Long (2000): *Smart simulation using collaborative formal and simulation engines*. In: *Proc. 2000 IEEE/ACM Int. Conf. on Computer-aided design*, ICCAD '00, IEEE Press, doi:10.1109/ICCAD.2000.896461.

[10] A. Kanade, R. Alur, F. Ivancic, S. Ramesh, S. Sankaranarayanan & K. C. Shashidhar (2009): *Generating and Analyzing Symbolic Traces of Simulink/Stateflow Models*. In: *Proc. 21st Int. Conf. Computer Aided Verification, CAV*, doi:10.1007/978-3-642-02658-4_33.

[11] G. Kreisselmeier & T. Birkholzer (1994): *Numerical nonlinear regulator design*. *Automatic Control, IEEE Transactions on* 39(1), doi:10.1109/9.273337.

[12] T. Mancini, F. Mari, A. Massini, I. Melatti, F. Merli & E. Tronci (2013): *System Level Formal Verification via Model Checking Driven Simulation*. In: *Computer Aided Verification - 25th International Conference, CAV*, doi:10.1007/978-3-642-39799-8_21.

[13] T. Mancini, F. Mari, A. Massini, I. Melatti & E. Tronci (2014): *Anytime System Level Verification via Random Exhaustive Hardware in the Loop Simulation*. In: *17th Euromicro Conference on Digital System Design, DSD*, doi:10.1109/DSD.2014.91.

[14] T. Mancini, F. Mari, A. Massini, I. Melatti & E. Tronci (2014): *System Level Formal Verification via Distributed Multi-core Hardware in the Loop Simulation*. In: *22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP*, doi:10.1109/PDP.2014.32.

[15] T. Mancini, F. Mari, A. Massini, I. Melatti & E. Tronci (2015): *SyLVaaS: System Level Formal Verification as a Service*. In: *23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP*, doi:10.1109/PDP.2015.119.

[16] T. Mancini, F. Mari, I. Melatti, I. Salvo, E. Tronci, J. K. Gruber, B. Hayes, M. Prodanovic & L. Elmegaard (2014): *Demand-aware price policy synthesis and verification services for Smart Grids*. In: *2014 IEEE International Conference on Smart Grid Communications, SmartGridComm*, doi:10.1109/SmartGridComm.2014.7007745.

[17] T. Mancini, E. Tronci, I. Salvo, F. Mari, A. Massini & I. Melatti (2015): *Computing Biological Model Parameters by Parallel Statistical Model Checking*. In: *Proc. Third Int. Conf. Bioinformatics and Biomedical Engineering, IWBBIO*, doi:10.1007/978-3-319-16480-9_52.

[18] B. Meenakshi, A. Bhatnagar & S. Roy (2006): *Tool for Translating Simulink Models into Input Language of a Model Checker*. In: *Proc. 8th Int. Conf. on Formal Engineering Methods, ICFEM*, doi:10.1007/11901433_33.

[19] N. Miskov-Zivanov, P. Zuliani, E. M. Clarke & J. R. Faeder (2013): *Studies of biological networks with statistical model checking: application to immune system cells*. In: *ACM Conference on Bioinformatics, Computational Biology and Biomedical Informatics. ACM-BCB*, doi:10.1145/2506583.2512390.

[20] K. Nanshi & F. Somenzi (2006): *Guiding simulation with increasingly refined abstract traces*. In: *Proc. 43rd annual Design Automation Conference*, DAC '06, ACM, New York, NY, USA, doi:10.1145/1146909.1147097.

[21] E.D. Sontag (1998): *Mathematical Control Theory: Deterministic Finite Dimensional Systems*. Texts in Applied Mathematics, Springer, doi:10.1007/978-1-4612-0577-7.

[22] S. Tripakis, C. Sofronis, P. Caspi & A. Curic (2005): *Translating discrete-time simulink to lustre*. *ACM Trans. Embedded Comput. Syst.* 4(4), doi:10.1145/1113830.1113834.

[23] E. Tronci, T. Mancini, I. Salvo, S. Sinisi, F. Mari, I. Melatti, A. Massini, F. Davi, T. Dierkes, R. Ehrig, S. Röblitz, B. Leeners, T. H. C. Kruger, M. Egli & F. Ille (2014): *Patient-specific models from inter-patient biological models and clinical records*. In: *Formal Methods in Computer-Aided Design, FMCAD*, doi:10.1109/FMCAD.2014.6987615.

[24] R. Venkatesh, U. Shrotri, P. Darke & P. Bokil (2012): *Test generation for large automotive models*. In: *IEEE Int. Conf. on Industrial Technology (ICIT)*, doi:10.1109/ICIT.2012.6210014.

[25] M. W. Whalen, D. D. Cofer, S. P. Miller, B. H. Krogh & W. Storm (2007): *Integration of Formal Analysis into a Model-Based Software Development Process*. In: *Proc. 12th Int. Workshop Formal Methods for Industrial Critical Systems, FMICS*, doi:10.1007/978-3-540-79707-4_7.

[26] C. H. Yang & D. L. Dill (1998): *Validation with guided search of the state space*. In: *Proc. 35th annual Design Automation Conference*, DAC '98, ACM, New York, NY, USA, doi:10.1145/277044.277201.

[27] P. Zuliani, A. Platzer & E. M. Clarke (2010): *Bayesian statistical model checking with application to Simulink/Stateflow verification*. In: *Proc. 13th ACM Int. Conf. on Hybrid Systems: Computation and Control, HSCC*, doi:10.1145/1755952.1755987.

[28] P. Zuliani, A. Platzer & E. M. Clarke (2013): *Bayesian statistical model checking with application to Stateflow/Simulink verification*. *Formal Methods in System Design* 43(2), doi:10.1007/s10703-013-0195-3.